CHAPTER 13

# Understanding Server Administration

## IN THIS CHAPTER

Administering Linux servers

Communicating with servers over networks

Setting up logging locally and remotely

Monitoring server systems

Managing servers in the enterprise

Although some system administration tasks are needed even on a desktop system (installing software, setting up printers, and so on), many new tasks appear when you set up a Linux system to act as a server. That's especially true if the server that you configure is made public to anyone on the Internet, where you can be overloaded with requests from good guys while needing to be constantly on guard against attacks from bad guys.

Dozens of different kinds of servers are available for Linux systems. Most servers serve up data to remote clients, but others serve the local system (such as those that gather logging messages or kick off maintenance tasks at set times using the `cron` facility). Many servers are represented by processes that run continuously in the background and respond to requests that come to them. These processes are referred to as *daemon* processes.

As the name implies, servers exist to serve. The data that they serve can include web pages, files, database information, email, and lots of other types of content. As a server administrator, some of the additional challenges to your system administration skills include the following:

**Remote access** To use a desktop system, you typically sit at its console. Server systems, by contrast, tend to be housed in racks in climate-controlled environments under lock and key. More often than not, after the physical computers are in place, most administration of those machines is done using remote access tools. Often, no graphical interface is available, so you must rely on command-line tools or browser-based interfaces to do things such as remote login, remote copying, and remote execution. The most common of these tools are built on the Secure Shell (SSH) facility.

**Diligent security** To be useful, a server must be able to accept requests for content from remote users and systems. Unlike desktop systems, which can simply close down all network

*Linux® Bible*, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

ports that allow incoming requests for access, a server must make itself vulnerable by allowing some access to its ports. That's why as a server administrator, it is important to open ports to services that are needed and lock down ports that are not needed. You can secure services using tools such as `iptables` and `firewalld` (firewall tools) and Security Enhanced Linux (to limit the resources a service can access from the local system).

**Continuous monitoring** Although you typically turn off your laptop or desktop system when you are not using it, servers usually stay on 24×7, 365 days a year. Because you don't want to sit next to each server and continuously monitor it personally, you can configure tools to monitor each server, gather log messages, and even forward suspicious messages to an email account of your choice. You can enable system activity reporters to gather data around the clock on CPU usage, memory usage, network activity, and disk access.

In this chapter, I lay out some of the basic tools and techniques that you need to know to administer remote Linux servers. You learn to use SSH tools to access your server securely, transfer data back and forth, and even launch remote desktops or graphical applications and have them appear on your local system. You learn to use remote logging and system activity reports to monitor system activities continuously.

## Starting with Server Administration

Whether you are installing a file server, web server, or any of the other server facilities available with Linux systems, many of the steps required for getting the server up and running are the same. Where server setup diverges is in the areas of configuration and tuning.

In later chapters, I describe specific servers and how they differ. In each of the server-related chapters that follow, you'll go through the same basic steps for getting that server started and available to be used by your clients.

### Step 1: Install the server

Although most server software is not preinstalled on the typical Linux system, any general-purpose Linux system offers the software packages needed to supply every major type of server available.

Sometimes, multiple software packages associated with a particular type of server are gathered together in package groups (sometimes called *package collections*). At other times, you just need to install the server packages you want individually. Here are some server package categories in Fedora and some of the packages available in each category:

**System logging server** The `rsyslog` service allows the local system to gather log messages delivered from a variety of components on the system. It can also act as a remote logging server, gathering logging messages sent from other logging servers. (The `rsyslog` service is described later in this chapter.) In recent Ubuntu, Fedora, and RHEL systems, log messages are gathered in the `systemd` journal, which can be picked up and redirected by the `rsyslog` service or displayed locally by the `journalctl` command.

**Print server** The Common UNIX Printing Service (`cups` package) is used most often to provide print server features on Linux systems. Packages that provide graphical administration of CUPS (`system-config-printer`) and printer drivers (`foomatic`, `hpijs`, and others) are also available when you install CUPS. (See Chapter 16, "Configuring a Print Server.")

**Web server** The Apache (`httpd` or `apache2` package) web server is the software used most often to serve web pages (HTTP content). Related packages include modules to help serve particular types of content (Perl, Python, PHP, and SSL connections). Likewise, there are packages of related documentation (`httpd-manual`), tools for monitoring web data (webalizer), and tools for providing web proxy services (squid). (See Chapter 17, "Configuring a Web Server.")

**FTP server** The Very Secure FTP daemon (`vsftpd` package) is the default FTP server used in Fedora and RHEL. Other FTP server packages include `proftpd` and `pure-ftpd`. (See Chapter 18, "Configuring an FTP Server.")

**Windows file server** Samba (`samba` package) allows a Linux system to act as a Windows file and print server. (See Chapter 19, "Configuring a Windows File Sharing [Samba] Server.")

**NFS file server** Network File System (NFS) is the standard Linux and UNIX feature for providing shared directories to other systems over a network. The `nfs-utils` package provides NFS services and related commands. (See Chapter 20, "Configuring an NFS File Server.")

**Mail server** These types of packages enable you to configure email servers, sometimes referred to as a Mail Transport Agent (MTA) server. You have several choices of email servers, including `sendmail`, `postfix` (default in Fedora and RHEL), and `exim`. Related packages, such as `dovecot`, allow the mail server to deliver email to clients.

**Directory server** Packages in this category provide remote and local authentication services. These include Kerberos (`krb5-server`), LDAP (`openldap-servers`), and NIS (`ypserv`).

**DNS server** The Berkeley Internet Name Domain service (`bind`) provides the software needed to configure a server to resolve hostnames into IP addresses.

**Network Time Protocol server** The `ntpd` or `chronyd` package provides a service that you can enable to sync up your system clock with clocks from public or private NTP servers.

**SQL server** The PostgreSQL (`postgresql` and `postgresql-server` packages) service is an object-relational database management system. Related packages provide PostgreSQL documentation and related tools. The MySQL (`mysql` and `mysql-server` packages) service is another popular open source SQL database server.

A community-developed branch of MySQL called MariaDB has supplanted MySQL on many Linux distributions.

## Step 2: Configure the server

Most server software packages are installed with a default configuration that leans more toward security than immediate full use. Here are some things to think about when you set out to configure a server.

### Using configuration files

Traditionally, Linux servers have been configured by editing plain-text files in the `/etc` directory (or subdirectories). Often, there is a primary configuration file; sometimes, there is a related configuration directory in which files ending in `.conf` can be pulled into the main configuration file.

The `httpd` package (Apache web server) is an example of a server package that has a primary configuration file and a directory where other configuration files can be dropped in and be included with the service. The main configuration file in Fedora and RHEL is `/etc/httpd/conf/httpd.conf`. The configuration directory is `/etc/httpd/conf.d/`.

After installing `httpd` and related packages, you will see files in the `/etc/httpd/conf.d/` directory that were placed there by different packages: `mod_ssl`, `mod_perl`, and so on. This is a way that add-on packages to a service can have their configuration information enabled in the `httpd` server, without the package trying to run a script to edit the main `httpd.conf` file.

The one downside to plain-text configuration files is that you don't get the kind of immediate error checking you get when you use graphical administration tools. You either have to run a test command (if the service includes one) or actually try to start the service to see if there is any problem with your configuration file.

**TIP**

**Instead of using `vi` to edit configuration files, use `vim`. Using the `vim` command can help you catch configuration file errors as you are editing.**

**The `vim` command knows about the formats of many configuration files (`passwd`, `httpd.conf`, `fstab`, and others). If you make a mistake and type an invalid term or option in one of those files, or break the format somehow, the color of the text changes. For example, in `/etc/fstab`, if you change the option `defaults` to `default`, the word's color changes.**

### Checking the default configuration

Most server software packages in Fedora and RHEL are installed with minimal configuration and lean more toward being secure than totally useful out of the box. While installing a software package, some Linux distributions ask you things such as the directory in which you want to install it or the user account with which you want to manage it.

Because RPM packages are designed to be installed unattended, the person installing the package has no choice on how it is installed. The files are installed in set locations, specific user accounts are enabled to manage it, and when you start the service, it might well offer limited accessibility. You are expected to configure the software after the package is installed to make the server fully functional.

Two examples of servers that are installed with limited functionality are mail servers (`sendmail` or `postfix` packages) and DNS servers (`bind` package). Both of these servers are installed with default configurations and start up on reboot. However, both also only listen for requests on your `localhost`. So, until you configure those servers, people who are not logged in to your local server cannot send mail to the mail server or use your computer as a public DNS server, respectively.

## Step 3: Start the server

Most services that you install in Linux are configured to start up when the system boots and then run continuously, listening for requests, until the system is shut down. There are two major facilities for managing services: `systemd` (used now by RHEL, Ubuntu, and Fedora) and SystemVinit scripts (used by Red Hat Enterprise Linux through RHEL 6.*x*).

Regardless of which facility is used on your Linux system, it is your job to do things such as set whether you want the service to come up when the system boots and to start, stop, and reload the service as needed (possibly to load new configuration files or temporarily stop access to the service). Commands for doing these tasks are described in Chapter 15, "Starting and Stopping Services."

Most, but not all, services are implemented as daemon processes. Here are a few things that you should know about those processes:

**User and group permissions** Daemon processes often run as users and groups other than root. For example, `httpd` runs as apache and `ntpd` runs as the ntp user. The reason for this is that if someone cracks these daemons, they would not have permissions to access files beyond what the services can access.

**Daemon configuration files** Often, a service has a configuration file for the daemon stored in the `/etc/sysconfig` directory. This is different than the service configuration file in that its job is often just to pass arguments to the server process itself rather than configure the service. For example, options you set in the `/etc/sysconfig/rsyslogd` file are passed to the `rsyslogd` daemon when it starts up. You can tell the daemon, for example, to output additional debugging information or accept remote logging messages. See the man page for the service (for example, `man rsyslogd`) to see what options are supported.

**Port numbers** Packets of data go to and from your system over network interfaces through ports for each supported protocol (usually UDP or TCP). Most standard services have specific port numbers to which daemons listen and to which clients connect. Unless you are trying to hide the location of a service, you typically don't change the ports on which a daemon process listens. When you go to secure a service, you must make sure that the port to the service is open on the firewall (see Chapter 25, "Securing Linux on a Network," for information on `iptables` and `firewalld` firewalls). Also, if you change a port on which the service is listening, and SELinux is in enforcing mode, SELinux may prevent the daemon from listening on that port (see Chapter 24, "Enhancing Linux Security with SELinux," for more information on SELinux).

**NOTE**

One reason for changing port numbers on a service is "security by obscurity." For example, the `sshd` service is a well-known target for people trying to break into a system by guessing logins and passwords on TCP port 22.

I have heard about people changing their Internet-facing `sshd` service to listen on some other port number (perhaps some unused, very high port number). Then they tell their friends or colleagues to log in to their machine from SSH by pointing to this other port. The idea is that port scanners looking to break into a system might be less likely to scan the normally unused port.

Not all services run continuously as daemon processes. Some older UNIX services ran on demand using the `xinetd` super server. Other services just run once on startup and exit. Still others run only a set number of times, being launched when the `crond` daemon sees that the service was configured to run at the particular time.

In recent years, previous `xinetd` services in RHEL and Fedora, such as `telnet` and `tftp`, have been converted to `systemd` services. A number of services, including `cockpit`, use `systemd` sockets to achieve the same results.

## Step 4: Secure the server

Opening your system to allow remote users to access it over the network is not a decision to be taken lightly. Crackers all over the world run programs to scan for vulnerable servers that they can take over for their data or their processing power. Luckily, there are measures that you can put in place on Linux systems to protect your servers and services from attacks and abuse.

Some common security techniques are described in the following sections. These and other topics are covered in more depth in Part V, "Learning Linux Security Techniques."

### Password protection

Good passwords and password policies are the first line of defense in protecting a Linux system. If someone can log in to your server via ssh as the root user with a password of `foobar`, expect to be cracked. A good technique is to disallow direct login by root and require every user to log in as a regular user and then use `su` or `sudo` to become root.

You can also use the Pluggable Authentication Module (PAM) facility to adjust the number of times that someone can have failed login attempts before blocking access to that person. PAM also includes other features for locking down authentication to your Linux server. For a description of PAM, see Chapter 23, "Understanding Advanced Linux Security."

Of course, you can bypass passwords altogether by requiring public key authentication. To use that type of authentication, you must make sure that any user you want to have access to your server has their public key copied to the server (such as through `ssh-copy-id`). Then they can use `ssh`, `scp`, or related commands to access that server without typing their password. See the section "Using key-based (passwordless) authentication" later in this chapter for further information.

### Firewalls

The `iptables` firewall service can track and respond to every packet coming from and going to network interfaces on your computer. Using `iptables`, you can drop or reject every packet making requests for services on your system except for those few that you have enabled. Further, you can tell `iptables` to allow service requests only from certain IP addresses (good guys) or not allow requests from other addresses (bad guys).

In recent RHEL and Fedora versions, the `firewalld` feature adds a layer of functionality to Linux firewall rules. With `firewalld`, you can not only insert firewall rules into the kernel, you can also organize firewall rules by dividing them up into zones and changing firewall rules on the fly to react to different events.

In each of the server chapters coming up, I describe what ports need to be open to allow access to services. Descriptions of how `iptables` and `firewalld` work are included in Chapter 25, "Securing Linux on a Network."

### TCP Wrappers

*TCP Wrappers*, which uses `/etc/hosts.allow` and `/etc/hosts.deny` files to allow and deny access in a variety of ways to selected services, was used primarily to secure older UNIX services, and it is no longer considered to be very secure. While the use of the TCP Wrapper program (`/usr/sbin/tcpd`) is only common on systems that use `xinetd`, the `/etc/hosts.allow` and `/etc/hosts.deny` files that the TCP Wrapper program checked before granting access to network services are often checked by daemons that are configured to do so. The configuration option within the configuration files for these daemons is often labeled as TCP Wrapper support.

### SELinux

Fedora, Red Hat Enterprise Linux, and other Linux distributions come with the Security Enhanced Linux (SELinux) feature included and in Enforcing mode. Although the default targeted mode doesn't have much impact on most applications that you run in Linux, it has a major impact on most major services.

A major function of SELinux is to protect the contents of your Linux system from the processes running on the system. In other words, SELinux makes sure a web server, FTP server, Samba server, or DNS server can access only a restricted set of files on the system (as defined by file contexts) and allows only a restricted set of features (as defined by Booleans and limited port access).

Details about how to use SELinux are contained in Chapter 24, "Enhancing Linux Security with SELinux."

### Security settings in configuration files

Within the configuration files of most services are values that you can set to secure the service further. For example, for file servers and web servers, you can restrict access to certain files or data based on username, hostname, IP address of the client, or other attributes.

## Step 5: Monitor the server

Because you can't be there to monitor every service, every minute, you need to put monitoring tools in place to watch your servers for you and make it easy for you to find out when something needs attention. Some of the tools that you can use to monitor your servers are described in the sections that follow.

### Configure logging

Using the `rsyslog` service (`rsyslogd` daemon), you can gather critical information and error conditions into log files about many different services. By default, in RHEL log messages from applications are directed into log files in the `/var/log` directory. For added security and convenience, log messages can also be directed to a centralized server, providing a single location to view and manage logging for a group of systems.

Several different software packages are available to work with `rsyslog` and manage log messages. The `logwatch` feature scans your log files each night and sends critical information gathered from those files to an email account of your choice. The `logrotate` feature backs up log files into compressed archives when the logs reach a certain size or pass a set amount of time since the previous backup.

The features for configuring and managing system logging are described in the section "Configuring System Logging" later in this chapter.

### Run system activity reports

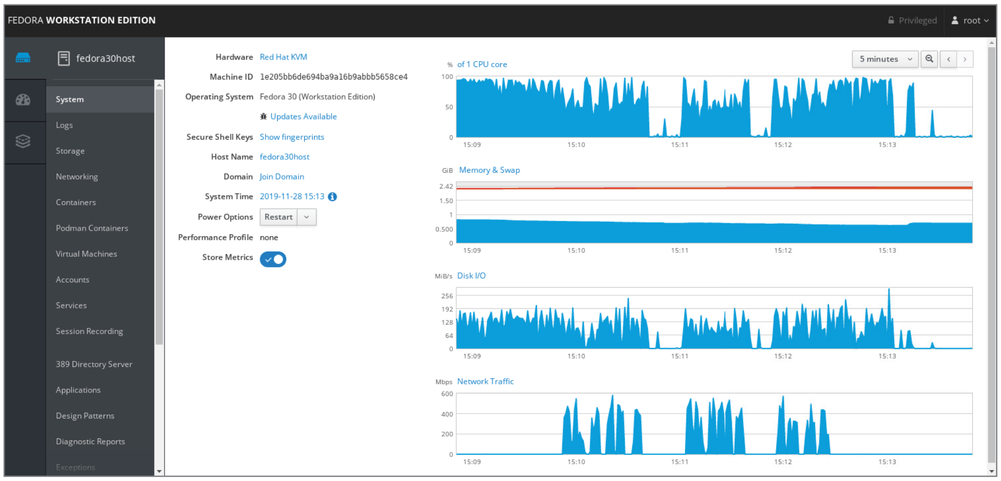
The `sar` facility (which is enabled by the `sysstat` package) can be configured to watch activities on your system such as memory usage, CPU usage, disk latency, network activities, and other resource drains. By default, the `sar` facility launches the `sadc` program every few minutes, day and night, to gather data. Viewing that data later can help you go back and figure out where and when demand is spiking on your system. The `sar` facility is described in the section "Checking System Resources with `sar`" later in this chapter.

### Watch activity live with Cockpit

With Cockpit running on your system, you can watch system activity in real time. Open your web browser to display the Cockpit console (`https://localhost:9090`). In real time, you can watch percentage of CPU use, memory and swap consumption, how much data is written to and from disk (disk i/o), and network traffic as it is gathered and displayed across the screen. Figure 13.1 shows an example of the System area of the Cockpit console, displaying activity data.

**FIGURE 13.1**

Log in to Cockpit



| | |
|---|---|
| Hardware | Red Hat KVM |
| Machine ID | 1e205bb6de694ba9a16b9abbb5658ce4 |
| Operating System | Fedora 30 (Workstation Edition) |
| | Updates Available |
| Secure Shell Keys | Show fingerprints |
| Host Name | fedora30host |
| Domain | Join Domain |
| System Time | 2019-11-28 15:13 |
| Power Options | Restart |
| Performance Profile | none |
| Store Metrics | ✓ |

## Keep system software up to date

As security holes are discovered and patched, you must make sure that the updated software packages containing those patches are installed on your servers. Again, with mission-critical servers, the safest and most efficient way is to use subscribed Red Hat Enterprise Linux systems for your servers and then deploy security-related package updates to your system as soon as they are released and tested.

To keep your personal server and desktop systems up to date, there are various graphical tools to add software and to check for updates. You can also use the `yum` command to check for and install all packages that are available for your RHEL or Fedora systems (enter **`dnf update`** or **`yum update`**).

## Check the filesystem for signs of crackers

To check your filesystem for possible intrusion, you can run commands such as `rpm -V` to check to see if any commands, document files, or configuration files have been tampered with on your system. For more information on `rpm -V`, refer to the description of `rpm -V` in Chapter 10, "Getting and Managing Software."

Now that you have an overview of how Linux server configuration is done, the next sections of this chapter focus on the tools that you need to access, secure, and maintain your Linux server systems.

## Checking and Setting Servers

If you are tasked with managing a Linux server, the following sections include a bunch of items that you can check. Keep in mind that nowadays many servers in large data centers are deployed and managed by larger platforms. So, know how the server is managed before you make any changes to it. Your changes might be overwritten automatically if you changed the defined state of that system.

## Managing Remote Access with the Secure Shell Service

The *Secure Shell tools* are a set of client and server applications that allow you to do basic communications between client computers and your Linux server. The tools include `ssh`, `scp`, `sftp`, and many others. Because communication is encrypted between the server and the clients, these tools are more secure than similar, older tools. For example, instead of using older remote login commands such as `telnet` or `rlogin`, you could use `ssh`. The `ssh` command can also replace older remote execution commands, such as `rsh`. Remote copy commands, such as `rcp`, can be replaced with secure commands such as `scp` and `rsync`.

With Secure Shell tools, both the authentication process and all communications that follow are encrypted. Communications from `telnet` and the older "`r`" commands expose passwords and all data to someone sniffing the network. Today, `telnet` and similar commands should be used only for testing access to remote ports, providing public services such as PXE booting, or doing other tasks that don't expose your private data.

**NOTE**

For a deeper discussion of encryption techniques, refer to Chapter 23, "Understanding Advanced Linux Security."

Most Linux systems include secure shell clients, and many include the secure shell server as well. If you are using the Fedora or RHEL distribution, for example, the client and server software packages that contain the `ssh` tools are `openssh`, `openssh-clients`, and `openssh-server` packages as follows:

```
# yum list installed | grep openssh
...
openssh.x86_64           7.9p1-5.fc30    @anaconda
openssh-clients.x86_64   7.9p1-5.fc30    @anaconda
openssh-server.x86_64    7.9p1-5.fc30    @anaconda
```

On Ubuntu, only the `openssh-clients` package is installed. It includes the functionality of the `openssh` package. If you need the server installed, use the `sudo apt-get install openssh-server` command.

```
$ sudo dpkg --list | grep openssh
openssh-client/bionic-updates,bionic-security,now 1:7.6p1-4ubuntu0.3 amd64
[installed]
  secure shell (SSH) client, for secure access to remote machines
openssh-client-ssh1/bionic 1:7.5p1-10 amd64
  secure shell (SSH) client for legacy SSH1 protocol

openssh-sftp-server/bionic-updates,bionic-security,now 1:7.6p1-4ubuntu0.3
amd64 [installed]
  secure shell (SSH) sftp server module, for SFTP access from remote machines
$ sudo apt-get install openssh-server
```

## Starting the openssh-server service

Linux systems that come with the `openssh-server` package already installed sometimes are not configured for it to start automatically. Managing Linux services (see Chapter 15, "Starting and Stopping Services") can be very different depending on the different distributions. Table 13.1 shows the commands to use in order to ensure that the `ssh` server daemon, `sshd`, is up and running on a Linux system.

**TABLE 13.1 Commands to Determine `sshd` Status**

| Distribution | Command to Determine `sshd` Status |
|---|---|
| RHEL 6 | `chkconfig --list sshd` |
| Fedora and RHEL 7 or later | `systemctl status sshd.service` |
| Ubuntu | `systemctl status ssh.service` |

If `sshd` is not currently running, you can start it by issuing one of the commands listed in Table 13.2. These commands need root privileges in order to work.

**TABLE 13.2 Commands to Start `sshd`**

| Distribution | Command to Start `sshd` |
|---|---|
| RHEL 6 | `service sshd start` |
| Fedora and RHEL 7 or later | `systemctl start sshd.service` |
| Ubuntu | `systemctl start ssh.service` |

The commands in Table 13.2 only start the `ssh` or `sshd` service. They do not configure it to start automatically at boot. To make sure the server service is set up to start automatically, you need to use one of the commands in Table 13.3 using root privileges.

**TABLE 13.3 Commands to Start `sshd` at Boot**

| Distribution | Command to Start `sshd` at Boot |
|---|---|
| RHEL 6 | `chkconfig sshd on` |
| Fedora and RHEL 7 or later | `systemctl enable sshd.service` |
| Ubuntu | `systemctl enable ssh.service` |

When you install `openssh-server` on Ubuntu, the `sshd` daemon is configured to start automatically at boot. Therefore, you may not need to run the command in Table 13.3 for your Ubuntu server.

Modify your firewall settings to allow the `openssh-client` to access port 22 (firewalls are covered in Chapter 25, "Securing Linux on a Network"). After the service is up and running and the firewall is properly configured, you should be able to use `ssh` client commands to access your system via the `ssh` server.

Any further configurations for what the `sshd` daemon is allowed to do are handled in the `/etc/ssh/sshd_config` file. At a minimum, set the `PermitRootLogin` setting to `no`. This stops anyone from remotely logging in as root.

```
# grep PermitRootLogin /etc/ssh/sshd_config
PermitRootLogin no
```

After you have changed the `sshd_config` file, restart the `sshd` service. After that point, if you use `ssh` to log in to that system from a remote client, you must do so as a regular user and then use `su` or `sudo` to become the root user.

## Using SSH client tools

Many tools for accessing remote Linux systems have been created to make use of the SSH service. The most frequently used of those tools is the `ssh` command, which can be used for remote login, remote execution, and other tasks. Commands such as `scp` and `rsync` can copy one or more files at a time between SSH client and server systems. The `sftp` command provides an FTP-like interface for traversing a remote filesystem and getting and putting files between the systems interactively.

By default, all of the SSH-related tools authenticate using standard Linux usernames and passwords, all done over encrypted connections. However, SSH also supports key-based authentication, which can be used to configure key-based and possibly passwordless authentication between clients and SSH servers, as described in the section "Using key-based (passwordless) authentication" later in this chapter.

### Using ssh for remote login

Use the `ssh` command from another Linux computer to test that you can log in to the Linux system running your `sshd` service. The `ssh` command is one that you will use often to access a shell on the servers you are configuring.

Try logging in to your Linux server from another Linux system using the ssh command. (If you don't have another Linux system, you can simulate this by typing localhost instead of the IP address and logging in as a local user.) The following is an example of remotely logging in to johndoe's account on 10.140.67.23:

```
$ ssh johndoe@10.140.67.23
The authenticity of host '10.140.67.23 (10.140.67.23)'
     can't be established.
RSA key fingerprint is
     a4:28:03:85:89:6d:08:fa:99:15:ed:fb:b0:67:55:89.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.140.67.23' (RSA) to the
     list of known hosts.
johndoe@10.140.67.23's password: *********
```

If this is the very first time that you have logged in to that remote system using the ssh command, the system asks you to confirm that you want to connect. Type yes, and press Enter. When prompted, enter the user's password.

When you type yes to continue, you accept the remote host's public key. At that point, the remote host's public key is downloaded to the client in the client's ~/.ssh/known_hosts file. Now data exchanged between these two systems can be encrypted and decrypted using RSA asymmetric encryption (see Chapter 23, "Understanding Advanced Linux Security"). After you are logged in to the remote system, you can begin typing shell commands. The connection functions like a normal login. The only difference is that the data is encrypted as it travels over the network.

13

When you are finished, type exit to end the remote connection. The connection is closed, and you are returned to the command prompt on your local system. (If the local shell doesn't return after you exit the remote shell, typing ~. usually closes the connection.)

```
$ exit
logout
Connection to 10.140.67.23 closed
```

After you have remotely connected to a system, a file in your local system subdirectory, ~.ssh/known_hosts, will exist. This file contains the public key of the remote host along with its IP address. Your server's public and private keys are stored in the /etc/ssh directory.

```
$ ls .ssh
known_hosts
$ cat .ssh/known_hosts
10.140.67.23 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAoyfJK1YwZhNmpHE4yLPZAZ9ZNEdRE7I159f3I
yGiH21Ijfqs
NYFR10ZlBLlYyTQi06r/9O19GwCaJ753InQ8FWHW+OOYOG5pQmghhn
/x0LD2uUb6egOu6zim1NEC
JwZf5DWkKdy4euCUEMSqADh/WYeuOSoZ0pp2IAVCdh6
w/PIHMF1HVR069cvdv+OTL4vD0X8llSpw
```

```
0ozqRptz2UQgQBBbBjK1RakD7fY1TrWv
NQhYG/ugt gPaY4JDYeY6OBzcadpxZmf7EYUw0ucXGVQ1a
NP/erIDOQ9rA0YNzCRv
y2LYCm2/9adpAxc+UYi5UsxTw4ewSBjmsXYq//Ahaw4mjw==
```

**TIP**

Any later attempts by this user to contact the server at 10.140.67.23 are authenticated using this stored key. If the server should change its key (which happens if the operating system is reinstalled or if keys are rotated), attempts to `ssh` to that system result in a refused connection and dire warnings that you may be under attack. If the key has indeed changed, in order to be able to `ssh` to that address again, just remove the host's key (the whole line) from your `known _hosts` file and you can copy over the new key.

### Using ssh for remote execution

Besides logging into a remote shell, the `ssh` command can be used to execute a command on the remote system and have the output returned to the local system. Here is an example:

```
$ ssh johndoe@10.140.67.23 hostname
johndoe@10.140.67.23's password: **********
host01.example.com
```

In the example just shown, the `hostname` command runs as the user `johndoe` on the Linux system located at IP address `10.140.67.23`. The output of the command is the name of the remote host (in this case, `host01.example.com`), which appears on the local screen.

If you run a remote execution command with `ssh` that includes options or arguments, be sure to surround the whole remote command line in quotes. Keep in mind that if you refer to files or directories in your remote commands, relative paths are interpreted in relation to the user's home directory, as shown here:

```
$ ssh johndoe@10.140.67.23 "cat myfile"
johndoe@10.140.67.23's password: **********
Contents of the myfile file located in johndoe's home directory.
```

The `ssh` command just shown goes to the remote host located at `10.140.67.23` and runs the `cat myfile` command as the user `johndoe`. This causes the contents of the `myfile` file from that system to be displayed on the local screen.

Another type of remote execution that you can do with `ssh` is X11 forwarding. If X11 forwarding is enabled on the server (`X11Forwarding yes` is set in the `/etc/sshd/sshd_config` file), you can run graphical applications from the server securely over the SSH connection using `ssh -X`. For a new server administrator, this means that if there are graphical administration tools installed on a server, you can run those tools without having to sit at the console, as in this example:

```
$ ssh -X johndoe@10.140.67.23 system-config-printer
johndoe@10.140.67.23's password: **********
```

After running this command, you are prompted for the root password. After that, the Printers window appears, ready for you to configure a printer. Just close the window when you are finished, and the local prompt returns. You can do this for any graphical administration tool or just regular X applications (such as the gedit graphical editor, so that you don't have to use vi).

If you want to run several X commands and don't want to have to reconnect each time, you can use X11 forwarding directly from a remote shell as well. Put them in the background and you can have several remote X applications running on your local desktop at once. Here's an example:

```
$ ssh -X johndoe@10.140.67.23
johndoe@10.140.67.23's password: *********
$ system-config-printer &
$ gedit &
$ exit
```

After you have finished using the graphical applications, close them as you would normally. Then type exit, as shown in the preceding code, to leave the remote shell and return to your local shell.

### Copying files between systems with scp and rsync

The scp command is similar to the old UNIX rcp command for copying files to and from Linux systems, except that all communications are encrypted. Files can be copied from the remote system to the local system or local to remote. You can also copy files recursively through a whole directory structure if you choose.

The following is an example of using the scp command to copy a file called memo from the home directory of the user chris to the /tmp directory on a remote computer as the user johndoe:

```
$ scp /home/chris/memo johndoe@10.140.67.23:/tmp
johndoe@10.140.67.23's password: ***************
memo       100%|****************|  153   0:00
```

You must enter the password for johndoe. After the password is accepted, the file is copied to the remote system successfully.

You can do recursive copies with scp using the -r option. Instead of a file, pass a directory name to the scp command and all files and directories below that point in the filesystem are copied to the other system.

```
$ scp johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
volname.1.gz                              100%  543   0.5KB/s  00:00
mtools.1.gz                               100% 6788   6.6KB/s  00:00
roqet.1.gz                                100% 2496   2.4KB/s  00:00
...
```

As long as the user johndoe has access to the files and directories on the remote system and the local user can write to the target directory (both are true in this case), the directory structure from /usr/share/man/man1 down is copied to the local /tmp directory.

The scp command can be used to back up files and directories over a network. However, if you compare scp to the rsync command, you see that rsync (which also works over SSH connections) is a better backup tool. Try running the scp command shown previously to copy the man1 directory (you can simulate the command using localhost instead of the IP address if you only have one accessible Linux system). Now enter the following on the system to which you copied the files:

```
$ ls -l /usr/share/man/man1/batch* /tmp/man1/batch*
-rw-r--r--.1 johndoe johndoe 2628 Apr 15 15:32 /tmp/man1/batch.1.gz
lrwxrwxrwx.1 root root 7 Feb 14 17:49 /usr/share/man/man1/batch.1.gz
      -> at.1.gz
```

Next, run the scp command again and list the files once more:

```
$ scp johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
$ ls -l /usr/share/man/man1/batch* /tmp/man1/batch*
-rw-r--r--.1 johndoe johndoe 2628 Apr 15 15:40 /tmp/man1/batch.1.gz
lrwxrwxrwx.1 root root 7 Feb 14 17:49 /usr/share/man/man1/batch.1.gz
      -> at.1.gz
```

The output of those commands tells you a few things about how scp works:

**Attributes lost** Permissions or date/time stamp attributes were not retained when the files were copied. If you were using scp as a backup tool, you would probably want to keep permissions and time stamps on the files if you needed to restore the files later.

**Symbolic links lost** The batch.1.gz file is actually a symbolic link to the at.1.gz file. Instead of copying the link, scp follows the link and actually copies the file. Again, if you were to restore this directory, batch.1.gz would be replaced by the actual at.1.gz file instead of a link to it.

**Copy repeated unnecessarily** If you watched the second scp output, you would notice that all files were copied again, even though the exact files being copied were already on the target. The updated modification date confirms this. By contrast, the rsync command can determine that a file has already been copied and not copy the file again.

The rsync command is a better network backup tool because it can overcome some of the shortcomings of scp just listed. Try running an rsync command to do the same action that scp just did, but with a few added options:

```
$ rm -rf /tmp/man1/
$ rsync -avl johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
sending incremental file list
man1/
```

```
man1/HEAD.1.gz
man1/Mail.1.gz -> mailx.1.gz
...
$ rsync -avl johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
sending incremental file list
sent 42362 bytes  received 13 bytes  9416.67 bytes/sec
total size is 7322223  speedup is 172.80
$ ls -l /usr/share/man/man1/batch* /tmp/man1/batch*
lrwxrwxrwx.1 johndoe johndoe 7 Feb 14 17:49 /tmp/man1/batch.1.gz
       -> at.1.gz
lrwxrwxrwx.1 root root 7 Feb 14 17:49 /usr/share/man/man1/batch.1.gz
       -> at.1.gz
```

After removing the /tmp/man1 directory, you run an rsync command to copy all of the files to the /tmp/man1 directory, using -a (recursive archive), -v (verbose), and -l (copy symbolic links). Then run the command immediately again and notice that nothing is copied. The rsync command knows that all of the files are there already, so it doesn't copy them again. This can be a tremendous savings of network bandwidth for directories with gigabytes of files where only a few megabytes change.

Also notice from the output of ls -l that the symbolic links have been preserved on the batch.1.gz file and so has the date/time stamp on the file. If you need to restore those files later, you can put them back exactly as they were.

This use of rsync is good for backups. But what if you wanted to mirror two directories, making the contents of two directory structures exactly the same on two machines? The following commands illustrate how to create an exact mirror of the directory structure on both machines using the directories shown with the previous rsync commands.

First, on the remote system, copy a new file into the directory being copied:

```
# cp /etc/services /usr/share/man/man1
```

Next, on the local system, run rsync to copy across any new files (in this case, just the directory and the new file, services):

```
$ rsync -avl johndoe@10.140.67.23:/usr/share/man/man1 /tmp
johndoe@10.140.67.23's password:
***************
sending incremental file list
man1/
man1/services
```

After that, go back to the remote system and remove the new file:

```
$ sudo rm /usr/share/man/man1/services
```

Now, on the local system, run rsync again and notice that nothing happens. At this point, the remote and local directories are different because the local system has the services file and the remote doesn't. That is correct behavior for a backup directory. (You want to have files on the backup in case something was removed by mistake.) However, if you want the remote and local directories to be mirrored, you would have to add the `--delete` option. The result is that the services file is deleted on the local system, making the remote and local directory structures in sync.

```
$ rsync -avl /usr/share/man/man1 localhost:/tmp
johndoe@10.140.67.23's password: ***************
sending incremental file list
man1/
$ rsync -avl --delete johndoe@10.140.67.23:/usr/share/man/man1 /tmp
johndoe@10.140.67.23's password: ***************
sending incremental file list
deleting man1/services
```

#### Interactive copying with sftp

If you don't know exactly what you want to copy to or from a remote system, you can use the `sftp` command to create an interactive FTP-style session over the SSH service. Using `sftp`, you can connect to a remote system over SSH, change directories, list directory contents, and then (given proper permission) get files from and put files on the server. Keep in mind that, despite its name, `sftp` has nothing to do with the FTP protocol and doesn't use FTP servers. It simply uses an FTP style of interaction between a client and a `sshd` server.

The following example shows the user `johndoe` connecting to `jd.example.com`:

```
$ sftp johndoe@jd.example.com
Connecting to jd.example.com
johndoe@jd.example.com's password: ***************
sftp>
```

At this point, you can begin an interactive FTP session. You can use `get` and `put` commands on files as you would with any FTP client, but with the comfort of knowing that you are working on an encrypted and secure connection. Because the FTP protocol passes usernames, passwords, and data in clear text, using `sftp` over SSH, if possible, is a much better alternative for allowing your users to copy files interactively from the system.

### Using key-based (passwordless) authentication

If you are using SSH tools to connect to the same systems throughout the day, you might find it inconvenient to be entering your password over and over again. Instead of using password-based authentication, SSH allows you to set up key-based authentication to use instead. Here's how it works:

- You create a public key and a private key.
- You guard the private key but copy the public key across to the user account on the remote host to which you want to do key-based authentication.
- With your key copied to the proper location, you use any SSH tools to connect to the user account on the remote host, but instead of asking you for a password, the remote SSH service compares the public key and the private key and allows you access if the two keys match.

When you create the keys, you are given the option to add a passphrase to your private key. If you decide to add a passphrase, even though you don't need to enter a password to authenticate to the remote system, you still need to enter your passphrase to unlock your private key. If you don't add a passphrase, you can communicate using your public/private key pairs in a way that is completely passwordless. However, if someone should get ahold of your private key, they could act as you in any communication that required that key.

The following procedure demonstrates how a local user named `chris` can set up key-based authentication to a remote user named `johndoe` at IP address `10.140.67.23`. If you don't have two Linux systems, you can simulate this by using two user accounts on your local system. I start by logging in as the local user named `chris` and typing the following to generate my local public/private key pair:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/chris/.ssh/id_rsa): ENTER
Enter passphrase (empty for no passphrase): ENTER
Enter same passphrase again: ENTER
Your identification has been saved in /home/chris/.ssh/id_rsa.
Your public key has been saved in /home/chris/.ssh/id_rsa.pub.
The key fingerprint is:
bf:06:f8:12:7f:f4:c3:0a:3a:01:7f:df:25:71:ec:1d chris@abc.example.com
The key's randomart image is:
 ...
```

I accepted the default RSA key (DSA keys are also allowed) and pressed Enter twice to have a blank passphrase associated with the key. As a result, my private key (`id_rsa`) and public key (`id_rsa.pub`) are copied to the `.ssh` directory in my local home directory. The next step is to copy that key over to a remote user so that I can use key-based authentication each time I connect to that user account with `ssh` tools:

```
 $ ssh-copy-id -i ~/.ssh/id_rsa.pub johndoe@10.140.67.23
johndoe@10.140.67.23's password:
***************
```

When prompted, I entered `johndoe`'s password. With that accepted, the public key belonging to `chris` is copied to the `authorized_keys` file in `johndoe`'s `.ssh` directory on the remote system. Now, the next time `chris` tries to connect to `johndoe`'s account, the SSH connection is authenticated using those keys. Because no passphrase is put on the private key, no passphrase is required to unlock that key when it is used.

Log into the machine with `ssh johndoe@10.140.67.23`, and check in the `$HOME/.ssh/authorized_keys` to make sure that you haven't added extra keys that you weren't expecting.

```
[chris]$ ssh johndoe@10.140.67.23
Last login: Sun Apr 17 10:12:22 2016 from  10.140.67.22
[johndoe]$
```

With the keys in place, chris could now use ssh, scp, rsync, or any other SSH-enabled command to do key-based authentication. Using these keys, for example, an rsync command could go into a cron script and automatically back up johndoe's home directory every night.

Want to secure your remote system further? After you have the keys in place on your remote system for everyone you want to allow to log in to that system, you can set the sshd service on the remote system to not allow password authentication by changing the PasswordAuthentication setting in the /etc/ssh/sshd_config file to no, so that it appears as follows:

```
PasswordAuthentication no
```

Then restart the sshd service (systemctl restart sshd). After that, anyone with a valid key is still accepted. Anyone who tries to log in without a key gets the following failure message and doesn't even get a chance to enter a username and password:

```
Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
```

## Configuring System Logging

With the knowledge of how to access your remote server using SSH tools, you can log in to the server and set up some of the services needed to make sure that it's running smoothly. System logging is one of the basic services configured for Linux to keep track of what is happening on the system.

The rsyslog service (rsyslogd daemon) provides the features to gather log messages from software running on the Linux system and direct those messages to local log files, devices, or remote logging hosts. Configuration of rsyslog is similar to the configuration of its predecessor, syslog. However, rsyslog allows you to add modules to manage and direct log messages more specifically.

In recent Red Hat Enterprise Linux and Fedora releases, the rsyslog facility leverages messages that are gathered and stored in the systemd journal. To display journal log messages directly from the systemd journal, instead of viewing them from files in the /var/log directory, use the journalctl command.

### Enabling system logging with rsyslog

Most of the files in the /var/log directory are populated with log messages directed to them from the rsyslog service. The rsyslogd daemon is the system logging daemon. It accepts log messages from a variety of other programs and writes them to the appropriate log files. This is better than having every program write directly to its own log file because it enables you to manage centrally how log files are handled.

Configuring rsyslogd to record varying levels of detail in the log files is possible. It can be told to ignore all but the most critical messages, or it can record every detail.

The rsyslogd daemon can even accept messages from other computers on your network. This remote logging feature is particularly handy because it enables you to centralize the management and review of the log files from many systems on your network. There is also a major security benefit to this practice.

With remote logging, if a system on your network is broken into, the cracker cannot delete or modify the log files because those files are stored on a separate computer. It is important to remember, however, that those log messages are not, by default, encrypted (though encryption can be enabled). Anyone tapping into your local network can eavesdrop on those messages as they pass from one machine to another. Also, although crackers may not be able to change old log entries, they can affect the system such that any new log messages should not be trusted.

Running a dedicated loghost, a computer that serves no purpose other than to record log messages from other computers on the network, is not uncommon. Because this system runs no other services, it is unlikely that it will be broken into. This makes it nearly impossible for crackers to erase their tracks completely.

### Understanding the rsyslog.conf file

The /etc/rsyslog.conf file is the primary configuration file for the rsyslog service. In the /etc/rsyslog.conf file, a modules section lets you include or not include specific features in your rsyslog service. The following is an example of the modules section of /etc/rsyslog.conf in RHEL 8:

```
module(load="imuxsock"
    # provides support for local system logging (e.g. via logger command)
     SysSock.Use="off") # Turn off message reception via local log socket;
                        # local messages are retrieved through imjournal now.
module(load="imjournal"
    # provides access to the systemd journal
        StateFile="imjournal.state") # File to store the position in the journal
#module(load="imklog")
    # reads kernel messages (the same are read from journald)
#module(load="immark")
    # provides --MARK-- message capability

# Provides UDP syslog reception
# for parameters see http://www.rsyslog.com/doc/imudp.html
#module(load="imudp") # needs to be done just once
#input(type="imudp" port="514")

# Provides TCP syslog reception
# for parameters see http://www.rsyslog.com/doc/imtcp.html
#module(load="imtcp") # needs to be done just once
#input(type="imtcp" port="514")
```

Entries beginning with module(load= load the modules that follow. Modules that are currently disabled are preceded by a pound sign (#). The imjournal module lets rsyslog access the systemd journal. The imuxsock module is needed to accept messages from the local system. (It should not be commented out—preceded by a pound sign—unless you have a specific reason to do so.) The imklog module logs kernel messages.

Modules not enabled by default include the immark module, which allows --MARK-- messages to be logged (used to indicate that a service is alive). The imudp and imtcp modules and related port number entries are used to allow the rsyslog service to accept remote logging messages and are discussed in more detail in the section "Setting up and using a loghost with rsyslogd" later in this chapter.

Most of the work done in the /etc/rsyslog.conf configuration file involves modifying the RULES section. The following is an example of some of the rules in the RULES section of the /etc/rsyslog.conf file (note that in Ubuntu, you need to look in the /etc/rsyslog.d directory for this configuration information):

```
#### RULES ####
# Log all kernel messages to the console.

# Logging much else clutters up the screen.

#kern.*                                          /dev/console
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none       /var/log/messages
# The authpriv file has restricted access.
authpriv.*                                       /var/log/secure
# Log all the mail messages in one place.
mail.*                                           -/var/log/maillog
# Log cron stuff
cron.*                                           /var/log/cron
```

Rules entries come in two columns. In the left column are designations of what messages are matched; the right column shows where matched messages go. Messages are matched based on facility (mail, cron, kern, and so on) and priority (starting at debug, info, notice, and up to crit, alert, and emerg), separated by a dot (.). So mail.info matches all messages from the mail service that are info level and above.

As for where the messages go, most messages are directed to files in the /var/log directory. You can, however, direct messages to a device (such as /dev/console) or a remote loghost (such as @loghost.example.com). The at sign (@) indicates that the name that follows is the name of the loghost.

By default, logging is done only to local files in the /var/log directory. However, if you uncomment the kern.* entry, you can easily direct kernel messages of all levels to your computer's console screen.

The first working entry in the preceding example shows that info level messages from all services (*) are matched by that rule, with the exception of messages from mail, authpriv, and cron services (which are excluded with the word none). All of the matched messages are directed to the /var/log/messages file.

The mail, authpriv (authentication messages), and cron (cron facility messages) services each has its own log files, as listed in the columns to their right. To understand the format of those and other log files, the format of the /var/log/messages file is described next.

### Understanding the messages log file

Because of the many programs and services that record information to the messages log file, understanding the format of this file is important. You can get a good early warning of problems developing on your system by examining this file. Each line in the file is a single message recorded by some program or service. Here is a snippet of an actual messages log file:

```
Feb 25 11:04:32 toys network: Bringing up loopback:  succeeded
Feb 25 11:04:35 toys network: Bringing up interface eth0:  succeeded
Feb 25 13:01:14 toys vsftpd(pam_unix)[10565]: authentication failure;
      logname= uid=0 euid=0 tty= ruser= rhost=10.0.0.5  user=chris
Feb 25 14:44:24 toys su(pam_unix)[11439]: session opened for
      user root by chris(uid=500)
```

The default message format in the /var/log/messages file is divided into five main parts. This format is determined by the following entry in the /etc/rsyslog.conf file:

```
module(load="builtin:omfile" Template="RSYSLOG_TraditionalFileFormat")
```

When you view messages in files from the /var/log directory, from left to right, message parts are as follows:

- The date and time that the message was logged
- The name of the computer from which the message came
- The program or service name to which the message pertains
- The process number (enclosed in square brackets) of the program sending the message
- The actual text message

Take another look at the preceding file snippet. In the first two lines, you can see that the network was restarted. The next line shows that the user named chris tried and failed to get to the FTP server on this system from a computer at address 10.0.0.5. (He typed the wrong password and authentication failed.) The last line shows chris using the su command to become root user.

By occasionally reviewing the messages and secure files, you could catch a cracking attempt before it is successful. If you see an excessive number of connection attempts for a particular service, especially if they are coming from systems on the Internet, you may be under attack.

### Setting up and using a loghost with rsyslogd

To redirect your computer's log files to another computer's `rsyslogd`, you must make changes to both the local and remote `rsyslog` configuration file, `/etc/rsyslog.conf`. Become root using the `su -` command, and then open the `/etc/rsyslog.conf` file in a text editor (such as `vi`).

#### *On the client side*

To send the messages to another computer (the loghost) instead of a file, start by replacing the log file name with the `@` character followed by the name of the loghost. For example, to direct the output of messages that are being sent to the `messages`, `secure`, and `maillog` log files to a loghost as well, add the lines in bold to the messages file:

```
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;news.none;authpriv.none;cron.none  /var/log/messages
*.info;mail.none;news.none;authpriv.none;cron.none  @loghost
# The authpriv file has restricted access.
authpriv.*                                        /var/log/secure
authpriv.*                                        @loghost
# Log all the mail messages in one place.
mail.*                                             -/var/log/maillog
mail.*                                            @loghost
```

The messages are now sent to the `rsyslogd` running on the computer named loghost. The name `loghost` was not an arbitrary choice. Creating such a hostname and making it an alias to the actual system acting as the loghost is customary. That way, if you ever need to switch the loghost duties to a different machine, you need to change only the loghost alias; you do not need to re-edit the `syslog.conf` file on every computer.

#### *On the loghost side*

The loghost that is set to accept the messages must listen for those messages on standard ports (514 UDP, although it can be configured to accept messages on 514 TCP as well). Here is how you would configure the Linux loghost that is also running the `rsyslog` service:

- Edit the `/etc/rsyslog.conf` file on the loghost system and uncomment the lines that enable the `rsyslogd` daemon to listen for remote log messages. Uncomment the first two lines to enable incoming UDP log messages on port 514 (default); uncomment the two lines after that to allow messages that use TCP protocol (also port 514):

  ```
  module(load="imudp") # needs to be done just once
  input(type="imudp" port="514")
  module(load="imtcp") # needs to be done just once
  input(type="imtcp" port="514")
  ```

- Open your firewall to allow new messages to be directed to your loghost. (See Chapter 25, "Securing Linux on a Network," for a description of how to open specific ports to allow access to your system.)
- Restart the `rsyslog` service (**`service rsyslog restart`** or **`systemctl restart rsyslog.service`**).
- If the service is running, you should be able to see that the service is listening on the ports that you enabled (UDP and/or TCP ports 514). Run the `netstat` command as follows to see that the `rsyslogd` daemon is listening on IPv4 and IPv6 ports 514 for both UDP and TCP services:

```
 # netstat -tupln | grep 514
tcp      0    0 0.0.0.0:514   0.0.0.0:*   LISTEN     25341/rsyslogd
tcp      0    0 :::514        :::*        LISTEN     25341/rsyslogd
udp      0    0 0.0.0.0:514   0.0.0.0:*              25341/rsyslogd
udp      0    0 :::514        :::*                   25341/rsyslogd
```

## Watching logs with logwatch

The `logwatch` service runs in most Linux systems that do system logging with `rsyslog`. Because logs on busy systems can become very large over time, it doesn't take long for there to be too many messages for a system administrator to watch every message in every log. To install the `logwatch` facility, enter the following:

```
# yum install logwatch
```

What `logwatch` does is gather messages once each night that look like they might represent a problem, put them in an email message, and send it to any email address the administrator chooses. To enable `logwatch`, all you have to do is install the `logwatch` package.

The `logwatch` service runs from a `cron` job (`0logwatch`) placed in `/etc/cron.daily`. The `/etc/logwatch/conf/logwatch.conf` file holds local settings. The default options used to gather log messages are set in the `/usr/share/logwatch/default.conf/logwatch.conf` file.

Some of the default settings define the location of log files (`/var/log`), location of the temporary directory (`/var/cache/logwatch`), and the recipient of the daily `logwatch` email (the local `root` user). Unless you expect to log in to the server to read `logwatch` messages, you probably want to change the `MailTo` setting in the `/etc/logwatch/conf/logwatch.conf` file:

```
MailTo = chris@example.com
```

Look in `/usr/share/logwatch/default.conf/logwatch.conf` for other settings to change (such as detail level or the time range for each report). Then make your additions to `/etc/logwatch/conf/logwatch.conf` as mentioned.

When the service is enabled (which it is just by installing the logwatch package), you will see a message each night in the root user's mailbox. When you are logged in as root, you can use the old mail command to view the root user's mailbox:

```
# mail
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 2 messages 2 new
>N  1 logwatch@abc.ex  Sun Feb 15 04:02 45/664   "Logwatch for abc"
    2 logwatch@abc.ex  Mon Feb 16 04:02 45/664   "Logwatch for abc"
& 1
& x
```

In mail, you should see email messages from logwatch run each day (here at 4:02 a.m.). Type the number of the message that you want to view and page through it with the spacebar or line by line by pressing Enter. Type **x** to exit when you are finished.

The kind of information that you see includes kernel errors, installed packages, authentication failures, and malfunctioning services. Disk space usage is reported, so you can see if your storage is filling up. Just by glancing through this logwatch message, you should get an idea whether sustained attacks are under way or if some repeated failures are taking place.

## Checking System Resources with sar

The System Activity Reporter (sar) is one of the oldest system monitoring facilities created for early UNIX systems—predating Linux by a few decades. The sar command itself can display system activity continuously, at set intervals (every second or two), and display it on the screen. It can also display system activity data that was gathered earlier.

The sar command is part of the sysstat package. When you install sysstat and enable the sysstat service, your system immediately begins gathering system activity data that can be reviewed later using certain options to the sar command.

```
# systemclt enable sysstat
# systemctl start sysstat
```

To read the data in the /var/log/sa/sa?? files, you can use some of the following sar commands:

```
# sar -u | less
Linux 5.3.8-200.fc30.x86_64 (fedora30host) 11/28/2019   _x86_64_  (1 CPU)

23:27:46     LINUX RESTART (1 CPU)

11:30:05 PM  CPU      %user   %nice %system    %iowait    %steal      %idle
11:40:06 PM  all       0.90    0.00    1.81       1.44      0.28      95.57
Average:     all       0.90    0.00    1.81       1.44      0.28      95.57
```

The -u option shows CPU usage. By default, the output starts at midnight on the current day and then shows how much processing time is being consumed by different parts of the system. The output continues to show the activity every 10 minutes until the current time is reached.

To see disk activity output, run the sar -d command. Again, output comes in 10-minute intervals starting at midnight.

```
# sar -d | less
Linux 5.3.8-200.fc30.x86_64 (fedora30host)  11/28/2019 _x86_64_ (1 CPU)

23:27:46     LINUX RESTART  (1 CPU)

11:30:05 PM        DEV   tps   rkB/s   wkB/s   areq-sz    aqu-sz  await...
11:40:06 PM     dev8-0 49.31 5663.94   50.38    115.89      0.03   1.00
11:40:06 PM   dev253-0 48.99 5664.09    7.38    115.78      0.05   0.98
11:40:06 PM   dev253-1 10.84    0.01   43.34      4.00      0.04   3.29
Average:        dev8-0 49.31 5663.94   50.38    115.89      0.03   1.00
Average:      dev253-0 48.99 5664.09    7.38    115.78      0.05   0.98
Average:      dev253-1 10.84    0.01   43.34      4.00      0.04   3.29
```

If you want to run sar activity reports live, you can do that by adding counts and time intervals to the command line, as shown here:

```
# sar -n DEV 5 2
Linux 5.3.8-200.fc30.x86_64 (fedora30host)  11/28/2019  _x86_64_  (1 CPU)
11:19:36 PM IFACE rxpck/s txpck/s  rxkB/s  txkB/s rxcmp/s txcmp/s...
11:19:41 PM    lo    5.42    5.42    1.06    1.06    0.00    0.00...
11:19:41 PM  ens3    0.00    0.00    0.00    0.00    0.00    0.00...
...
Average: IFACE rxpck/s txpck/s rxkB/s txkB/ rxcmp/s txcmp/s rxmcst/s
Average:    lo    7.21    7.21   1.42  1.42    0.00    0.00     0.00
Average:  ens3    0.00    0.00   0.00  0.00    0.00    0.00     0.00
Average: wlan0    4.70    4.00   4.81  0.63    0.00    0.00     0.00

Average:  pan0    0.00    0.00   0.00  0.00    0.00    0.00     0.00
Average:  tun0    3.70    2.90   4.42  0.19    0.00    0.00     0.00
```

With the -n Dev example just shown, you can see how much activity came across the different network interfaces on your system. You can see how many packets were transmitted and received and how many KB of data were transmitted and received. In that example, samplings of data were taken every 5 seconds and repeated twice.

Refer to the sar, sadc, sa1, and sa2 man pages for more information on how sar data can be gathered and displayed.

## Checking System Space

Although logwatch can give you a daily snapshot of space consumption on your system disks, the df and du commands can help you immediately see how much disk space is available. The following sections show examples of those commands.

### Displaying system space with df

You can display the space available in your filesystems using the df command. To see the amount of space available on all of the mounted filesystems on your Linux computer, type df with no options:

```
$ df
Filesystem  1k-blocks     Used  Available  Use%   Mounted on
/dev/sda3    30645460  2958356   26130408   11%   /
/dev/sda2       46668     8340      35919   19%   /boot
...
```

This example output shows the space available on the hard disk partition mounted on the / (root) directory (/dev/sda1) and /boot partition (/dev/sda2). Disk space is shown in 1KB blocks. To produce output in a more human-readable form, use the -h option:

```
$ df -h
Filesystem            Size  Used  Avail  Use%   Mounted on
/dev/sda3              29G  2.9G    24G   11%   /
/dev/sda2              46M  8.2M    25M   19%   /boot
...
```

With the df -h option, output appears in a friendlier megabyte or gigabyte listing. Other options with df enable you to do the following:

- Print only filesystems of a particular type (-t type).
- Exclude filesystems of a particular type (-x type). For example, type df -x tmpfs -x devtmpfs to exclude temporary filesystem types (limiting output to filesystems that represent real storage areas).
- Include filesystems that have no space, such as /proc and /dev/pts (-a).
- List only available and used inodes (-i).
- Display disk space in certain block sizes (--block-size=#).

### Checking disk usage with du

To find out how much space is being consumed by a particular directory (and its subdirectories), use the du command. With no options, du lists all directories below the current directory, along with the space consumed by each directory. At the end, du produces total disk space used within that directory structure.

The du command is a good way to check how much space is being used by a particular user (du /home/jake) or in a particular filesystem partition (du /var). By default, disk space is displayed in 1KB block sizes. To make the output friendlier (in kilobytes, megabytes, and gigabytes), use the `-h` option as follows:

```
$ du -h /home/jake
114k    /home/jake/httpd/stuff
234k    /home/jake/httpd
137k    /home/jake/uucp/data
701k    /home/jake/uucp
1.0M    /home/jake
```

The output shows the disk space used in each directory under the home directory of the user named `jake` (`/home/jake`). Disk space consumed is shown in kilobytes (`k`) and megabytes (`M`). The total space consumed by `/home/jake` is shown on the last line. Add the `–s` option to see total disk space used for a directory and its subdirectories.

## Finding disk consumption with find

The `find` command is a great way to find file consumption of your hard disk using a variety of criteria. You can get a good idea of where disk space can be recovered by finding files that are over a certain size or were created by a particular person.

**NOTE**

You must be the root user to run this command effectively, unless you are just checking your personal files. If you are not the root user, there are many places in the filesystem for which you do not have permission to check. Regular users can usually check their own home directories but not those of others.

In the following example, the `find` command searches the root filesystem (`/`) for any files owned by the user named `jake` (`-user jake`) and prints the filenames. The output of the `find` command is organized in a long listing in size order (`ls -ldS`). Finally, that output is sent to the file `/tmp/jake`. When you view the file `/tmp/jake` (for example, `less /tmp/jake`), you will find all of the files that are owned by the user `jake` listed in size order. Here is the command line:

```
# find / -xdev -user jake -print | xargs ls -ldS > /tmp/jake
```

**TIP**

The `-xdev` option prevents filesystems other than the selected filesystem from being searched. This is a good way to cut out lots of junk that may be output from the `/proc` filesystem. It can also keep large, remotely mounted filesystems from being searched.

Here's another example, except that instead of looking for a user's files, we're looking for files larger than 100 kilobytes (`-size +100M`):

```
# find / -xdev -size +100M | xargs ls -ldS > /tmp/size
```

You can save yourself lots of disk space just by removing some of the largest files that are no longer needed. In this example, you can see that large files are sorted by size in the `/tmp/size` file.

## Managing Servers in the Enterprise

Most of the server configuration covered in this book describes how to install systems manually and work directly on host computers. Having to set up each host individually would be far too inefficient for modern data centers consisting of dozens, hundreds, or even thousands of computers. To make the process of setting up Linux servers in a large data center more efficient, some of the following are employed:

**Automated deployments** One way to install systems without having to step through a manual install process is with PXE booting. By setting up a PXE server and booting a computer on that network from a PXE-enabled network interface card, you can start a full install of that system simply by booting the system. Once the install is done, the system can reboot to run from the installed system.

**Generic host systems** By making your host systems as generic as possible, individual installation, configuration, and upgrades can be greatly simplified. This can be automated in layers, where the base system is installed by PXE booting, configuration is done through features such as cloud-int, and applications can bring along their own dependencies when they run. On the application level, this can be done by running an application from inside a virtual machine or container. When the application is done running, it can be discarded without leaving its dependent software on the host.

**Separation of management and worker systems** Instead of individually managing host systems, a separate platform can offer a way to manage large sets of systems. To do this, a platform such as OpenStack or OpenShift can have management nodes (in some cases called *control plane* or *master nodes*) manage the machines where the workload actually runs (sometimes called *workers*, *slaves*, or just *nodes*). This separation of tasks by host type makes it possible to have applications deployed on any available worker that meets the needs of the application (such as available memory or CPU).

Keep in mind that understanding how individual applications are configured and services are run is still the foundation for these more advanced ways of managing data center resources. Although in-depth coverage of enterprise deployment and monitoring tools is outside the scope of this book, refer to Part VI, "Engaging with Cloud Computing," for an introduction to how different Linux-based cloud platforms manage these issues.

## Summary

Although many different types of servers are available with Linux systems, the basic procedure for installing and configuring a server is essentially the same. The normal course of events is to install, configure, start, secure, and monitor your servers. Basic tasks that apply to all servers include using networking tools (particularly SSH tools) to log in, copy files, or execute remote commands.

Because an administrator can't be logged in watching servers all the time, tools for gathering data and reviewing the log data later are very important when administering Linux servers. The `rsyslog` facility can be used for local and remote logging. The `sar` facility gathers live data or plays back data gathered earlier at 10-minute intervals. Cockpit lets you watch CPU, memory, disk, and networking activity live from a web user interface. To watch disk space, you can run `df` and `du` commands.

The skills described in this chapter are designed to help you build a foundation to do enterprise-quality system administration in the future. Although these skills are useful, to manage many Linux systems at the same time, you need to extend your skills by using automating deployment and monitoring tools, as described in the cloud computing section of this book.

Although it is easy to set up networking to reach your servers in simple, default cases, more complex network configuration requires a knowledge of networking configuration files and related tools. The next chapter describes how to set up and administer networking in Linux.

## Exercises

The exercises in this section cover some of the basic tools for connecting to and watching over your Linux servers. As usual, you can accomplish the tasks here in several ways. So, don't worry if you don't go about the exercises in the same way as shown in the answers, as long as you get the same results. If you are stuck, solutions to the tasks are shown in Appendix B.

Some of the exercises assume that you have a second Linux system available that you can log in to and try different commands. On that second system, you need to make sure that the `sshd` service is running, that the firewall is open, and that `ssh` is allowed for the user account that you are trying to log in to (root is often blocked by `sshd`).

If you have only one Linux system, you can create an additional user account and simply simulate communications with another system by connecting to the name `localhost` instead, as shown in this example:

```
# useradd joe
# passwd joe
# ssh joe@localhost
```

1. Using the `ssh` command, log in to another computer (or the local computer) using any account to which you have access. Enter the password when prompted.
2. Using remote execution with the `ssh` command, display the contents of a remote `/etc/system-release` file and have its contents displayed on the local system.

3. Use the `ssh` command to use X11 forwarding to display a `gedit` window on your local system; then save a file in the remote user's home directory.
4. Recursively copy all of the files from the `/usr/share/selinux` directory on a remote system to the `/tmp` directory on your local system in such a way that all of the modification times on the files are updated to the time on the local system when they are copied.
5. Recursively copy all of the files from the `/usr/share/logwatch` directory on a remote system to the `/tmp` directory on your local system in such a way that all of the modification times on the files from the remote system are maintained on the local system.
6. Create a public/private key pair to use for SSH communications (no passphrase on the key), copy the public key file to a remote user's account with `ssh-copy-id`, and use key-based authentication to log in to that user account without having to enter a password.
7. Create an entry in `/etc/rsyslog.conf` that stores all authentication messages (`authpriv`) info level and higher into a file named `/var/log/myauth`. From one terminal, watch the file as data comes into it, and in another terminal, try to `ssh` into your local machine as any valid user with a bad password.
8. Use the `du` command to determine the largest directory structures under `/usr/share`, sort them from largest to smallest, and list the top ten of those directories in terms of size.
9. Use the `df` command to show the space that is used and available from all of the filesystems currently attached to the local system but exclude any `tmpfs` or `devtmpfs` filesystems.
10. Find any files in the `/usr` directory that are more that 10MB in size.